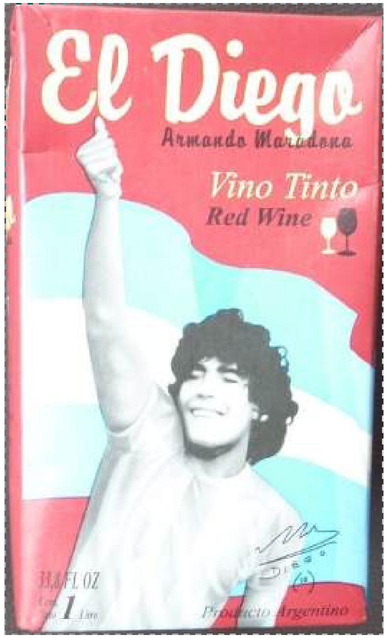
El Diego
Armando Maradona
Vino Tinto
Red Wine
Producto Argentino

Índice

1. Algoritmos

#include <algorithm> #include <numeric>

Algo	Params	Funcion
sort, stable_sort	f, l	ordena el intervalo
nth_element	f, nth, l	*void* ordena el n-esimo, y particiona el resto
fill, fill_n	f, l / n, elem	*void* llena [f, l) o [f, f+n) con elem
lower_bound, upper_bound	f, l, elem	*it* al primer / ultimo donde se puede insertar elem para que quede ordenada
binary_search	f, l, elem	*bool* esta elem en [f, l)
copy	f, l, resul	hace resul+i=f+i $\forall i$
find, find_if, find_first_of	f, l, elem / pred / f2, l2	*it* encuentra i $\in$[f,l) tq. i=elem, pred(i), i$\in$[f2,l2)
count, count_if	f, l, elem/pred	cuenta elem, pred(i)
search	f, l, f2, l2	busca [f2,l2) $\in$ [f,l)
replace, replace_if	f, l, old / pred, new	cambia old / pred(i) por new
reverse	f, l	da vuelta
partition, stable_partition	f, l, pred	pred(i) ad, !pred(i) atras
min_element, max_element	f, l, [comp]	*it* min, max de [f,l]
lexicographical_compare	f1,l1,f2,l2	*bool* con [f1,l1]¡[f2,l2]
next/prev_permutation	f,l	deja en [f,l) la perm sig, ant
set_intersection, set_difference, set_union, set_symmetric_difference,	f1, l1, f2, l2, res	[res, ...) la op. de conj
push_heap, pop_heap, make_heap	f, l, e / e /	mete/saca e en heap [f,l), hace un heap de [f,l)
is_heap	f,l	*bool* es [f,l) un heap
accumulate	f,l,i,[op]	$T = \sum$/oper de [f,l)
inner_product	f1, l1, f2, i	T = i + [f1, l1) . [f2, ...)
partial_sum	f, l, r, [op]	r+i = $\sum$/oper de [f,f+i] $\forall i \in$[f,l)
__builtin_ffs	unsigned int x	Pos. del primer 1 desde la derecha
__builtin_clz	unsigned int x	Cant. de ceros desde la izquierda.
__builtin_ctz	unsigned int x	Cant. de ceros desde la derecha.
__builtin_popcount	unsigned int x	Cant. de 1's en x.
__builtin_parity	unsigned int x	1 si x es par, 0 si es impar.

2. Estructuras

2.1. RMQ (static)

Dado un arreglo y una operación asociativa idempotente, get(i, j) opera sobre el rango [i, j). Restricción: LVL $\geq$ 2*ceil(logn); Usar [ ] para llenar arreglo y luego build().

```
struct RMQ{
  #define LVL 10
  tipo vec[LVL][1<<(LVL+1)];
  tipo &operator[](int p){return vec[0][p];}
  tipo get(int i, int j) {//intervalo [i,j)
    int p = 31-__builtin_clz(j-i);
    return min(vec[p][i],vec[p][j-(1<<p)]);
  }
  void build(int n) {//O(nlogn)
    int mp = 31-__builtin_clz(n);
    forn(p, mp) forn(x, n-(1<<p))
      vec[p+1][x] = min(vec[p][x], vec[p][x+(1<<p)]);
  }
};
```

2.2. RMQ (dynamic)

Dado un arreglo y una operación asociativa con neutro, get(i, j) opera sobre el rango [i, j).

```
#define MAXN 100000
struct RMQ{
  static const int sz=65536;//2*2^ceil(log(n))
  tipo t[4*MAXN];
  tipo &operator[](int p){return t[sz+p];}
  void init(int n){//O(nlgn)
    sz = 1 << (32-__builtin_clz(n));
    fill(t, t+2*sz, 0); // 0=elemento neutro
  }
  void updall(){//O(n)
    dforn(i, sz) t[i]=max(t[2*i], t[2*i+1]);}
  tipo get(int i, int j, int n=1, int a=0, int b=sz/2){//O(lgn)
    if(j<=a || i>=b) return 0;//neutro
    if(i<=a && b<=j) return t[n];
    int c=(a+b)/2;
    return max(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
  }
```

```
  void set(int p, tipo val){//O(lgn)
    for(p+=sz/2; p>0 && t[p]!=val;){
      t[p]=val;
      p/=2;
      val=max(t[p*2], t[p*2+1]);
    }
  }
}rmq;
//Usage:
cin >> n; rmq.init(n); forn(i, n) cin >> rmq[i]; rmq.updall();
```

2.3. Fenwick Tree

For 2D threat each column as a Fenwick tree, by adding a nested for in each operation

```
struct Fenwick{
  static const int sz=1000001;
  tipo t[sz];
  tipo sum(int a, int b){return sum(b)-sum(a-1);}
  void adjust(int p, tipo v){//valid with p in [1, sz), O(lgn)
    for(; p<sz; p+=(p&-p)) t[p]+=v; }
    tipo sum(int p){//cumulative sum in [1, p], O(lgn)
    tipo s=0;
    for(; p; p-=(p&-p)) s+=t[p];
    return s;
  }
  //get largest value with cumulative sum less than or equal to x;
  //for smallest, pass x-1 and add 1 to result
  int getind(tipo x) {//O(lgn)
      int idx = 0, mask = N;
      while(mask && idx < N) {
        int t = idx + mask;
      if(x >= tree[t])
          idx = t, x -= tree[t];
        mask >>= 1;
      }
      return idx;
  }
};
```

2.4. Union-Find

```
struct UnionFind{
  vector<int> f;//the array contains the parent of each node
  void init(int n){f.clear(); f.insert(f.begin(), n, -1);}
  int comp(int x){return (f[x]==-1?x:f[x]=comp(f[x]));}//O(1)
  void join(int i, int j) { if(comp(i)!=comp(j)) f[comp(i)] = comp(j); }
};
```

2.5. Disjoint Intervals

```
bool operator< (const ii &a, const ii &b) {return a.fst<b.fst;}
//Stores intervals as [first, second]
//in case of a collision it joins them in a single interval
struct disjoint_intervals {
  set<ii> segs;
  void insert(ii v) {//O(lgn)
    if(v.snd-v.fst==0.) return;//OJO
    set<ii>::iterator it,at;
    at = it = segs.lower_bound(v);
    if (at!=segs.begin() && (--at)->snd >= v.fst)
      v.fst = at->fst, --it;
    for(; it!=segs.end() && it->fst <= v.snd; segs.erase(it++))
      v.snd=max(v.snd, it->snd);
    segs.insert(v);
  }
};
```

2.6. RMQ (2D)

```
struct RMQ2D{
  static const int sz=1024;
  RMQ t[sz];
  RMQ &operator[](int p){return t[sz/2+p];}
  void build(int n, int m){//O(nm)
    forr(y, sz/2, sz/2+m)
      t[y].build(m);
    forr(y, sz/2+m, sz)
      forn(x, sz)
        t[y].t[x]=0;
    dforn(y, sz/2)
      forn(x, sz)
        t[y].t[x]=max(t[y*2].t[x], t[y*2+1].t[x]);
  }
  void set(int x, int y, tipo v){//O(lgm.lgn)
    y+=sz/2;
```

```
    t[y].set(x, v);
    while(y/=2)
      t[y].set(x, max(t[y*2][x], t[y*2+1][x]));
  }
  //O(lgm.lgn)
  int get(int x1, int y1, int x2, int y2, int n=1, int a=0, int b=sz/2){
    if(y2<=a || y1>=b) return 0;
    if(y1<=a && b<=y2) return t[n].get(x1, x2);
    int c=(a+b)/2;
    return max(get(x1, y1, x2, y2, 2*n, a, c),
         get(x1, y1, x2, y2, 2*n+1, c, b));
  }
};
//Example to initialize a grid of M rows and N columns:
RMQ2D rmq;
forn(i, M)
  forn(j, N)
    cin >> rmq[i][j];
rmq.build(N, M);
```

2.7. Big Int

```
#define BASEXP 6
#define BASE 1000000
#define LMAX 1000
struct bint{
    int l;
    tipo n[LMAX];
    bint(tipo x){
        l=0;
        forn(i, LMAX){
            n[i]=x%BASE;
            x/=BASE;
            l+=!!x||!i;
        }
    }
    bint(){THIS = bint(0);}
    bint(string x){
l=(x.size()-1)/BASEXP+1;
        fill(n, n+LMAX, 0);
        tipo r=1;
        forn(i, sz(x)){
            n[i / BASEXP] += r * (x[x.size()-1-i]-'0');
            r*=10; if(r==BASE)r=1;
        }
    }
    void out(){
    cout << n[l-1];
    dforn(i, l-1) printf("%6.6llu", n[i]);//6=BASEXP!
  }
  void invar(){
    fill(n+l, n+LMAX, 0);
    while(l>1 && !n[l-1]) l--;
  }
};
bint operator+(const bint&a, const bint&b){
  bint c;
    c.l = max(a.l, b.l);
    tipo q = 0;
    forn(i, c.l) q += a.n[i]+b.n[i], c.n[i]=q %BASE, q/=BASE;
    if(q) c.n[c.l++] = q;
    c.invar();
    return c;
}
pair<bint, bool> lresta(const bint& a, const bint& b)   // c = a - b
{
  bint c;
    c.l = max(a.l, b.l);
    tipo q = 0;
    forn(i, c.l) q += a.n[i]-b.n[i], c.n[i]=(q+BASE) %BASE, q=(q+BASE)/
        BASE-1;
    c.invar();
    return mkp(c, !q);
}
bint& operator-= (bint& a, const bint& b){return a=lresta(a, b).fst;}
bint operator- (const bint&a, const bint&b){return lresta(a, b).fst;}
bool operator< (const bint&a, const bint&b){return !lresta(a, b).snd;}
bool operator<= (const bint&a, const bint&b){return lresta(b, a).snd;}
bool operator==(const bint&a, const bint&b){return a <= b && b <= a;}
bint operator*(const bint&a, tipo b){
    bint c;
    tipo q = 0;
    forn(i, a.l) q += a.n[i]*b, c.n[i] = q %BASE, q/=BASE;
    c.l = a.l;
    while(q) c.n[c.l++] = q %BASE, q/=BASE;
    c.invar();
```

```
    return c;
}
bint operator*(const bint&a, const bint&b){
    bint c;
    c.l = a.l+b.l;
    fill(c.n, c.n+b.l, 0);
    forn(i, a.l){
        tipo q = 0;
        forn(j, b.l) q += a.n[i]*b.n[j]+c.n[i+j], c.n[i+j] = q %BASE, q /=BASE;
        c.n[i+b.l] = q;
    }
    c.invar();
    return c;
}
pair<bint, tipo> ldiv(const bint& a, tipo b){// c = a / b ; rm = a % b
bint c;
  tipo rm = 0;
dforn(i, a.l){
            rm = rm * BASE + a.n[i];
            c.n[i] = rm / b;
            rm %= b;
    }
    c.l = a.l;
    c.invar();
    return mkp(c, rm);
}
bint operator/(const bint&a, tipo b){return ldiv(a, b).fst;}
tipo operator%(const bint&a, tipo b){return ldiv(a, b).snd;}
pair<bint, bint> ldiv(const bint& a, const bint& b){
  bint c;
    bint rm = 0;
    dforn(i, a.l){
        if (rm.l==1 && !rm.n[0])
            rm.n[0] = a.n[i];
        else{
            dforn(j, rm.l) rm.n[j+1] = rm.n[j];
            rm.n[0] = a.n[i];
            rm.l++;
        }
        tipo q = rm.n[b.l] * BASE + rm.n[b.l-1];
        tipo u = q / (b.n[b.l-1] + 1);
        tipo v = q /  b.n[b.l-1] + 1;
        while (u < v-1){
            tipo m = (u+v)/2;
            if (b*m <= rm) u = m;
            else v = m;
        }
        c.n[i]=u;
        rm-=b*u;
    }
  c.l=a.l;
    c.invar();
    return mkp(c, rm);
}
bint operator/(const bint&a, const bint&b){return ldiv(a, b).fst;}
bint operator%(const bint&a, const bint&b){return ldiv(a, b).snd;}
```

2.8. Modnum

```
struct mnum{
  static const tipo mod=12582917;
  tipo v;
  mnum(tipo v=0): v(v%mod) {}
  mnum operator+(mnum b){return v+b.v;}
  mnum operator-(mnum b){return v>=b.v? v-b.v : mod-b.v+v;}
  mnum operator*(mnum b){return v*b.v;}
  mnum operator^(int n){
    if(!n) return 1;
    return n%2? (*this)^(n/2)*(this) : (*this)^(n/2);}
};
```

2.9. Bittrie

```
struct bitrie{
  static const int sz=1<<5;//5=ceil(log(n))
  int V;//valor del nodo
  vector<bitrie> ch;//childs
  bitrie():V(0){}//NEUTRO
  void set(int p, int v, int bit=sz>>1){//O(log sz)
    if(bit){
      ch.resize(2);
      ch[(p&bit)>0].set(p, v, bit>>1);
      V=max(ch[0].V, ch[1].V);
    }
    else V=v;
  }
```

```
  int get(int i, int j, int a=0, int b=sz){//O(log sz)
    if(j<=a || i>=b) return 0;//NEUTRO
    if(i<=a && b<=j) return V;
    if(!sz(ch)) return V;
    int c=(a+b)/2;
    return max(ch[0].get(i, j, a, c), ch[1].get(i, j, c, b));
  }
};
```

3. Strings

3.1. Trie

```
struct Trie{
  map<char, Trie> m;
  void add(char s[]){
    if(s[0]) m[s[i]].add(s+1);
  }
  void dfs(){
    //Do stuff
    forall(it, m)
      it->second.dfs();
  }
};
```

3.2. Suffix Array

```
#define MAX_N 1000
#define RABOUND(x) (x<n? RA[x] : 0)
//SA will hold the suffixes in order.
int SA[MAX_N], RA[MAX_N], n;
string s; //input string, n=sz(s)

void countingSort(int k){
  int f[MAX_N], tmpSA[MAX_N];
  zero(f);
  forn(i, n) f[RABOUND(i+k)]++;
  int sum=0;
  forn(i, max(255, n)){
    int t=f[i]; f[i]=sum; sum+=t;}
  forn(i, n)
    tmpSA[f[RABOUND(SA[i]+k)]++]=SA[i];
  memcpy(SA, tmpSA, sizeof(SA));
}
void constructSA(){//O(n log n)
  n=sz(s);
  forn(i, n) SA[i]=i, RA[i]=s[i];
  for(int k=1; k<n; k<<=1){
    countingSort(k), countingSort(0);
    int r, tmpRA[MAX_N];
    tmpRA[SA[0]]=r=0;
    forr(i, 1, n)
      tmpRA[SA[i]]=(RA[SA[i]]==RA[SA[i-1]] && RA[SA[i]+k]==RA[SA[i-1]+k]
          )? r : ++r;
    memcpy(RA, tmpRA, sizeof(RA));
    if(RA[SA[n-1]]==n-1) break;
  }
}
void print(){//for debug
  forn(i, n)
    cout << i << ' ' <<
    s.substr(SA[i], s.find( '$', SA[i])-SA[i]) << endl;}
```

3.3. String Matching With Suffix Array

```
//returns (lowerbound, upperbound) of the search
ii stringMatching(string P){ //O(sz(P)lgn)
  int lo=0, hi=n-1, mid=lo;
  while(lo<hi){
    mid=(lo+hi)/2;
    int res=s.compare(SA[mid], sz(P), P);
    if(res>=0) hi=mid;
    else lo=mid+1;
  }
  if(s.compare(SA[lo], sz(P), P)!=0) return ii(-1, -1);
  ii ans; ans.fst=lo;
  lo=0, hi=n-1, mid;
  while(lo<hi){
    mid=(lo+hi)/2;
    int res=s.compare(SA[mid], sz(P), P);
    if(res>0) hi=mid;
    else lo=mid+1;
  }
  if(s.compare(SA[hi], sz(P), P)!=0) hi--;
  ans.snd=hi;
  return ans;
```

```
}
```

3.4. LCP (Longest Common Prefix)

```
//Calculates the LCP between consecutives suffixes in the Suffix Array.
//LCP[i] is the length of the LCP between SA[i] and SA[i-1]
int LCP[MAX_N];
void computeLCP(){//O(n)
int phi[MAX_N], PLCP[MAX_N];
  phi[SA[0]]=-1;
  forr(i, 1, n) phi[SA[i]]=SA[i-1];
  int L=0;
  forn(i, n){
    if(phi[i]==-1) {PLCP[i]=0; continue;}
    while(s[i+L]==s[phi[i]+L]) L++;
    PLCP[i]=L;
    L=max(L-1, 0);
  }
  forn(i, n) LCP[i]=PLCP[SA[i]];
}
```

4. Geometría

#define EPS 1e-9

4.1. Punto

```
struct pto{
  tipo x, y;
  pto(tipo x=0, tipo y=0):x(x),y(y){}
  pto operator+(pto a){return pto(x+a.x, y+a.y);}
  pto operator-(pto a){return pto(x-a.x, y-a.y);}
  pto operator+(tipo a){return pto(x+a, y+a);}
  pto operator*(tipo a){return pto(x*a, y*a);}
  pto operator/(tipo a){return pto(x/a, y/a);}
  //dot product, producto interno:
  tipo operator*(pto a){return x*a.x+y*a.y;}
  //module of the cross product or vectorial product:
  //if a is less than 180 clockwise from b, a^b>0
  tipo operator^(pto a){return x*a.y-y*a.x;}
  //returns true if this is at the left side of line qr
  bool left(pto q, pto r){return ((q-*this)^(r-*this))>0;}
  bool operator<(const pto &a) const{return x<a.x || (abs(x-a.x)<EPS &&
      y<a.y);}
bool operator==(pto a){return abs(x-a.x)<EPS && abs(y-a.y)<EPS;}
  double norm(){return sqrt(x*x+y*y);}
  tipo norm_sq(){return x*x+y*y;}
};
double dist(pto a, pto b){return (b-a).norm();}
typedef pto vec;

double angle(pto a, pto o, pto b){
  vec oa=a-o, ob=b-o;
  return acos((oa*ob) / sqrt(oa.norm_sq()*ob.norm_sq()));}

//rotate p by theta rads CCW w.r.t. origin (0,0)
pto rotate(pto p, double theta){
  return pto(p.x*cos(theta)-p.y*sin(theta),
     p.x*sin(theta)+p.y*cos(theta));
}
```

4.2. Line

```
struct line{
  line() {}
  double a,b,c;//Ax+By=C
//pto MUST store float coordinates!
  line(double a, double b, double c):a(a),b(b),c(c){}
  line(pto p, pto q): a(q.y-p.y), b(p.x-q.x), c(a*p.x+b*p.y) {}
};
bool parallels(line l1, line l2){return abs(l1.a*l2.b-l2.a*l1.b)<EPS;}
pto inter(line l1, line l2){//intersection
  double det=l1.a*l2.b-l2.a*l1.b;
  if(abs(det)<EPS) return pto(INF, INF);//parallels
  return pto(l2.b*l1.c-l1.b*l2.c, l1.a*l2.c-l2.a*l1.c)/det;
}
```

4.3. Segment

```
struct segm{
  pto s,f;
  segm(pto s, pto f):s(s), f(f) {}
  pto closest(pto p) {//use for dist to point
     double l2 = dist_sq(s, f);
     if(l2==0.) return s;
     double t =((p-s)*(f-s))/l2;
```

```
    if (t<0.) return s;//not write if is a line
    else if(t>1.)return f;//not write if is a line
    return s+((f-s)*t);
  }
  bool inside(pto p){
return ((s-p)^(f-p))==0 && min(s, f)<*this&&*this<max(s, f);}
};

bool insidebox(pto a, pto b, pto p) {
  return (a.x-p.x)*(p.x-b.x)>-EPS && (a.y-p.y)*(p.y-b.y)>-EPS;
}
pto inter(segm s1, segm s2){
  pto r=inter(line(s1.s, s1.f), line(s2.s, s2.f));
  if(insidebox(s1.s,s1.f,p) && insidebox(s2.s,s2.f,p))
      return r;
  return pto(INF, INF);
}
```

4.4. Rectangle

```
struct rect{
  //lower-left and upper-right corners
  pto lw, up;
};
//returns if there's an intersection and stores it in r
bool inter(rect a, rect b, rect &r){
  r.lw=pto(max(a.lw.x, b.lw.x), max(a.lw.y, b.lw.y));
  r.up=pto(min(a.up.x, b.up.x), min(a.up.y, b.up.y));
//check case when only a edge is common
  return r.lw.x<r.up.x && r.lw.y<r.up.y;
}
```

4.5. Polygon Area

```
double area(vector<tipo> &p){//O(sz(p))
  double area=0;
  forn(i, sz(p)) area+=p[i]^p[(i+1)%sz(p)];
  //if points are in clockwise order then area is negative
  return abs(area)/2;
}
//Area ellipse = M_PI*a*b where a and b are the semi axis lengths
//Area triangle = sqrt(s*(s-a)(s-b)(s-c)) where s=(a+b+c)/2
```

4.6. Circle

```
vec perp(vec v){return vec(-v.y, v.x);}
line bisector(pto x, pto y){
  line l=line(x, y); pto m=(x+y)/2;
  return line(-l.b, l.a, -l.b*m.x+l.a*m.y);
}
struct Circle{
  pto o;
  double r;
//circle determined by three points, uses line
  Circle(pto x, pto y, pto z){
    o=inter(bisector(x, y), bisector(y, z));
    r=dist(o, x);
  }
  pair<pto, pto> ptosTang(pto p){
    pto m=(p+o)/2;
    tipo d=dist(o, m);
    tipo a=r*r/(2*d);
    tipo h=sqrt(r*r-a*a);
    pto m2=o+(m-o)*a/d;
    vec per=perp(m-o)/d;
    return mkp(m2-per*h, m2+per*h);
  }
};
//finds the center of the circle containing p1 and p2 with radius r
//as there may be two solutions swap p1, p2 to get the other
bool circle2PtsRad(pto p1, pto p2, double r, pto &c){
        double d2=(p1-p2).norm_sq(), det=r*r/d2-0.25;
        if(det<0) return false;
        c=(p1+p2)/2+perp(p2-p1)*sqrt(det);
        return true;
}
```

4.7. Point in Poly

```
//checks if v is inside of P, using ray casting
//works with convex and concave.
//excludes boundaries, handle it separately using segment.inside()
bool inPolygon(pto v, vector<pto>& P) {
  bool c = false;
  forn(i, sz(P)){
    int j=(i+1)%sz(P);
    if((P[j].y>v.y) != (P[i].y > v.y) &&
  (v.x < (P[i].x - P[j].x) * (v.y-P[j].y) / (P[i].y - P[j].y) + P[j].x))
```

```
      c = !c;
  }
  return c;
}
```

4.8. Convex Check CHECK

```
bool isConvex(vector<int> &p){//O(N)
  int N=sz(p);
  if(N<3) return false;
  bool isLeft=p[0].left(p[1], p[2]);
  forr(i, 1, N)
    if(p[i].left(p[(i+1)%N], p[(i+2)%N])!=isLeft)
      return false;
  return true; }
```

4.9. Convex Hull

```
//stores convex hull of P in S, CCW order
void CH(vector<pto>& P, vector<pto> &S){
  S.clear();
  sort(P.begin(), P.end());
  forn(i, sz(P)){
    while(sz(S)>= 2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back();
    S.pb(P[i]);
  }
  S.pop_back();
  int k=sz(S);
  dforn(i, sz(P)){
    while(sz(S) >= k+2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back
        ();
    S.pb(P[i]);
  }
  S.pop_back();
}
```

4.10. Cut Polygon

```
//cuts polygon Q along the line ab
//stores the left side (swap a, b for the right one) in P
void cutPolygon(pto a, pto b, vector<pto> Q, vector<pto> &P){
  P.clear();
  forn(i, sz(Q)){
    double left1=(b-a)^(Q[i]-a), left2=(b-a)^(Q[(i+1)%sz(Q)]-a);
    if(left1>=0) P.pb(Q[i]);
    if(left1*left2<0)
      P.pb(inter(line(Q[i], Q[(i+1)%sz(Q)]), line(a, b)));
  }
}
```

4.11. Bresenham

```
//plot a line approximation in a 2d map
void bresenham(pto a, pto b){
  pto d=b-a; d.x=abs(d.x), d.y=abs(d.y);
  pto s(a.x<b.x? 1: -1, a.y<b.y? 1: -1);
  int err=d.x-d.y;
  while(1){
    m[a.x][a.y]=1;//plot
    if(a==b) break;
    int e2=2*err;
    if(e2 > -d.y){
      err-=d.y, a.x+=s.x;
    if(e2 < d.x)
      err+= d.x, a.y+= s.y;
  }
}
```

4.12. Rotate Matrix

```
//rotates matrix t 90 degrees clockwise
//using auxiliary matrix t2(faster)
void rotate(){
  forn(x, n) forn(y, n)
    t2[n-y-1][x]=t[x][y];
  memcpy(t, t2, sizeof(t));
}
```

5. Math

5.1. GCD

```
tipo gcd(tipo a, tipo b){return a?gcd(b %a, a):b;}
```

5.2. LCM

```
tipo lcm(tipo a, tipo b){return a*b/gcd(a, b);}
```

5.3. Simpson

```
double integral(double a, double b, int n=10000) {//O(n), n=cantdiv
  double area=0, h=(b-a)/n, fa=f(a), fb;
  forn(i, n){
    fb=f(a+h*(i+1));
    area+=fa+ 4*f(a+h*(i+0.5)) +fb, fa=fb;
  }
  return area*h/6.;}
```

5.4. Fraction

```
struct frac{
  tipo p,q;
  frac(tipo p=0, tipo q=1):p(p),q(q) {norm();}
  tipo mcd(tipo a, tipo b){return a?mcd(b %a, a):b;}
  void norm(){
    tipo a = mcd(p,q);
    if(a) p/=a, q/=a;
    else q=1;
    if (q<0) q=-q, p=-p;}
  frac operator+(const frac& o){
    tipo a = mcd(q,o.q);
    return frac(p*(o.q/a)+o.p*(q/a), q*(o.q/a));}
  frac operator-(const frac& o){
    tipo a = mcd(q,o.q);
    return frac(p*(o.q/a)-o.p*(q/a), q*(o.q/a));}
  frac operator*(frac o){
    tipo a = mcd(q,o.q), b = mcd(o.q,p);
    return frac((p/b)*(o.p/a), (q/a)*(o.q/b));}
  frac operator/(frac o){
    tipo a = mcd(q,o.q), b = mcd(o.p,p);
    return frac((p/b)*(o.q/a),(q/a)*(o.p/b));}
  bool operator<(frac o){return (*this-o).p<0;}
  bool operator==(frac o){return p==o.p&&q==o.q;}
};
```

5.5. Polinomio

```
#define MAX_GR  20
struct poly {
  int p[MAX_GR];//guarda los coeficientes del polinomio
  poly(){zero(p);}
  int gr(){//calculates grade of the polynomial
    dforn(i,MAX_GR) if(p[i]) return i;
    return 0; }
  bool isnull() {return gr()==0 && !p[0];}
  poly operator+(poly b) {// - is analogous
    poly c=THIS;
    forn(i,MAX_GR) c.p[i]+=b.p[i];
    return c;
  }
  poly operator*(poly b) {
    poly c;
    forn(i,MAX_GR) forn(k,i+1) c.p[i]+=p[k]*b.p[i-k];
    return c;
  }
  int eval(int v) {
    int sum = 0;
    forn(i,MAX_GR) sum+=p[i]*pow(v,i);
    return sum;
  }
  //the following function generates the roots of the polynomial
//it can be easily modified to return float roots
  set<int> roots(){
    set<int> roots;
    int a0 = abs(p[0]), an = abs(p[gr()]);
    vector<int> ps,qs;
    forr(p,1,sqrt(a0)+1) if (a0%p==0) ps.pb(p),ps.pb(a0/p);
    forr(q,1,sqrt(an)+1) if (an%q==0) qs.pb(q),qs.pb(an/q);
    forall(pt,ps)
      forall(qt,qs) if ( (*pt) % (*qt)==0 ) {
        int root = abs((*pt) / (*qt));
        if (eval(root)==0) roots.insert(root);
      }
    return roots;
  }
};
//the following functions allows parsing an expression like
//34+150+4*45
//into a polynomial(el numero en funcion de la base)
#define LAST(s) (sz(s)? s[sz(s)-1] : 0)
#define POP(s) s.erase(--s.end());
poly D(string &s) {
  poly d;
  for(int i=0; isdigit(LAST(s)); i++) d.p[i]=LAST(s)-'0', POP(s);
  return d;}
```

```

poly T(string &s) {
  poly t=D(s);
  if (LAST(s)=='*'){POP(s); return T(s)*t;}
  return t;
}
//main function, call this to parse
poly E(string &s) {
  poly e=T(s);
  if (LAST(s)=='+'){POP(s); return E(s)+e;}
  return e;
}
```

6. Grafos

6.1. Dijkstra

```
#define INF 1e9
int N;
#define MAX_V 250001
vector<ii> G[MAX_V];
//To add an edge use
#define add(a, b, w) G[a].pb(mkp(w, b))

ll dijkstra(int s, int t){//O(|E| log |V|)
  priority_queue<ii, vector<ii>, greater<ii> > Q;
  vector<ll> dist(N, INF); vector<int> dad(N, -1);
  Q.push(mkp(0, s)); dist[s] = 0;
  while(sz(Q)){
    ii p = Q.top(); Q.pop();
    if(p.snd == t) break;
    forall(it, G[p.snd])
      if(dist[p.snd]+it->first < dist[it->snd]){
        dist[it->snd] = dist[p.snd] + it->fst;
        dad[it->snd] = p.snd;
        Q.push(mkp(dist[it->snd], it->snd));
      }
  }
  return dist[t];
  if(dist[t]<INF)//path generator
    for(int i=t; i!=-1; i=dad[i])
      printf("%d%c", i, (i==s?'\n':' '));
}
```

6.2. Bellman-Ford

```
vector<ii> G[MAX_N];//ady. list with pairs (weight, dst)
int dist[MAX_N];
void bford(int src){//O(VE)
  dist[src]=0;
  forn(i, N-1) forn(j, N) if(dist[j]!=INF) forall(it, G[j])
    dist[it->snd]=min(dist[it->snd], dist[j]+it->fst);
}

bool hasNegCycle(){
  forn(j, N) if(dist[j]!=INF) forall(it, G[j])
    if(dist[it->snd]>dist[j]+it->fst) return true;
  //inside if: all points reachable from it->snd will have -INF distance (do bfs)
  return false;
}
```

6.3. Floyd-Warshall

G[i][j] contains weight of edge (i, j) or INF G[i][i]=0

```
int G[MAX_N][MAX_N];
void floyd(){//O(N^3)
forn(k, N) forn(i, N) if(G[i][k]!=INF) forn(j, N) if(G[k][j]!=INF)
  G[i][j]=min(G[i][j], G[i][k]+G[k][j]);
}
bool inNegCycle(int v){
  return G[v][v]<0;}
//checks if there's a neg. cycle in path from a to b
bool hasNegCycle(int a, int b){
  forn(i, N) if(G[a][i]!=INF && G[i][i]<0 && G[i][b]!=INF)
    return true;
  return false;
}
```

6.4. 2-SAT + Tarjan SCC

We have a vertex representing a var and other for his negation. Every edge stored in G represents an implication. To add an equation of the form a——b, use addor(a, b) MAX=max cant var, n=cant var

```
#define addor(a, b) (G[neg(a)].pb(b), G[neg(b)].pb(a))
vector<int> G[MAX*2];
```

```
//idx[i]=index assigned in the dfs
//lw[i]=lowest index(closer from the root) reachable from i
int lw[MAX*2], idx[MAX*2], qidx;
stack<int> q;
int qcmp, cmp[MAX*2];
//verdad[cmp[i]]=valor de la variable i
bool verdad[MAX*2+1];

int neg(int x) { return x>=n? x-n : x+n;}
void tjn(int v){
  lw[v]=idx[v]=++qidx;
  q.push(v), cmp[v]=-2;
  forall(it, G[v]){
    if(!idx[*it] || cmp[*it]==-2){
      if(!idx[*it]) tjn(*it);
      lw[v]=min(lw[v], lw[*it]);
    }
  }
  if(lw[v]==idx[v]){
    qcmp++;
    int x;
    do{x=q.top(); q.pop(); cmp[x]=qcmp;}while(x!=v);
    verdad[qcmp]=(cmp[neg(v)]<0);
  }
}
//remember to CLEAR G!!!
bool satisf(){//O(n)
  memset(idx, 0, sizeof(idx)), qidx=0;
  memset(cmp, -1, sizeof(cmp)), qcmp=0;
  forn(i, n){
    if(!idx[i]) tjn(i);
    if(!idx[neg(i)]) tjn(neg(i));
  }
  forn(i, n) if(cmp[i]==cmp[neg(i)]) return false;
  return true;
}
```

6.5. Articulation Points

```
int N;
vector<int> G[1000000];
//V[i]=node number(if visited), L[i]= lowest V[i] reachable from i
int qV, V[1000000], L[1000000], P[1000000];
void dfs(int v, int f){
  L[v]=V[v]=++qV;
  forall(it, G[v])
    if(!V[*it]){
      dfs(*it, v);
      L[v] = min(L[v], L[*it]);
      P[v]+= L[*it]>=V[v];
    }
    else if(*it!=f)
      L[v]=min(L[v], V[*it]);
}
int cantart(){ //O(n)
  qV=0;
  zero(V), zero(P);
  dfs(1, 0); P[1]--;
  int q=0;
  forn(i, N) if(P[i]) q++;
return q;
}
```

6.6. LCA + Climb

```
LCA
#define POT2(x) (1<<(x))
//f[v][k] holds the 2^k father of v
//L[v] holds the level of v
int N, f[100001][20], L[100001];
void build(){//f[i][0] must be filled previously, O(nlgn)
  forn(k, 20-1) forn(i, N) f[i][k+1]=f[f[i][k]][k];}
int lg(int x){//=floor(log2(x))
  int i;
  for (i=0;(1<<i)<=x;i++) ;
  return i-1;
}
int climb(int a, int d){//O(lgn)
  if(!d) return a;
  dforn(i, lg(L[a])+1)
    if(POT2(i)<=d)
      a=f[a][i], d-=POT2(i);
    return a;
}
int lca(int a, int b){//O(lgn)
```

```
  if(L[a]<L[b]) swap(a, b);
  a=climb(a, L[a]-L[b]);
  if(a==b) return a;
  dforn(i, lg(L[a])+1)
    if(f[a][i]!=f[b][i])
      a=f[a][i], b=f[b][i];
  return f[a][0];
}
```

7. Network Flow

7.1. Edmonds Karp's

```
#define MAX_V 1000
#define INF 1e9
//special nodes
#define SRC 0
#define SNK 1
map<int, int> G[MAX_V];//limpiar esto
//To add an edge use
#define add(a, b, w) G[a][b]=w
int f, p[MAX_V];
void augment(int v, int minE){
  if(v==SRC) f=minE;
  else if(p[v]!=-1){
    augment(p[v], min(minE, G[p[v]][v]));
    G[p[v]][v]-=f, G[v][p[v]]+=f;
  }
}
ll maxflow(){//O(VE^2)
  ll Mf=0;
  do{
    f=0;
    char used[MAX_V]; queue<int> q; q.push(SRC);
    zero(used), memset(p, -1, sizeof(p));
    while(sz(q)){
      int u=q.front(); q.pop();
      if(u==SNK) break;
      forall(it, G[u])
        if(it->snd>0 && !used[it->fst])
          used[it->fst]=true, q.push(it->fst), p[it->fst]=u;
    }
    augment(SNK, INF);
    Mf+=f;
  }while(f);
  return Mf;
}
```

7.2. Push-Relabel

```
#define MAX_V 1000
int N;//valid nodes are [0...N-1]
#define INF 1e9
//special nodes
#define SRC 0
#define SNK 1
map<int, int> G[MAX_V];
//To add an edge use
#define add(a, b, w) G[a][b]=w
ll excess[MAX_V];
int height[MAX_V], active[MAX_V], count[2*MAX_V+1];
queue<int> Q;
void enqueue(int v) {
  if (!active[v] && excess[v] > 0) active[v]=true, Q.push(v); }
void push(int a, int b) {
  int amt = min(excess[a], ll(G[a][b]));
  if(height[a] <= height[b] || amt == 0) return;
  G[a][b]-=amt, G[b][a]+=amt;
  excess[b] += amt, excess[a] -= amt;
  enqueue(b);
}
void gap(int k) {
  forn(v, N){
    if (height[v] < k) continue;
    count[height[v]]--;
    height[v] = max(height[v], N+1);
    count[height[v]]++;
    enqueue(v);
  }
}
void relabel(int v) {
  count[height[v]]--;
  height[v] = 2*N;
  forall(it, G[v])
    if(it->snd)
      height[v] = min(height[v], height[it->fst] + 1);
```

```
  count[height[v]]++;
  enqueue(v);
}
ll maxflow() {//O(V^3)
  zero(height), zero(active), zero(count), zero(excess);
  count[0] = N-1;
  count[N] = 1;
  height[SRC] = N;
  active[SRC] = active[SNK] = true;
  forall(it, G[SRC]){
    excess[SRC] += it->snd;
    push(SRC, it->fst);
  }
  while(sz(Q)) {
    int v = Q.front(); Q.pop();
    active[v]=false;
  forall(it, G[v]) push(v, it->fst);
  if(excess[v] > 0)
    count[height[v]] == 1? gap(height[v]):relabel(v);
  }
  ll mf=0;
  forall(it, G[SRC]) mf+=G[it->fst][SRC];
  return mf;
}
```

8. Ayudamemoria

Límites

```
#include <climits> //INT_MIN, LONG_MAX, ULLONG_MAX, etc.
```

Cant. decimales

```
#include <iomanip>
cout << setprecision(2) << fixed;
```

Rellenar con espacios(para justificar)

```
#include <iomanip>
cout << setfill('␣') << setw(3) << 2 << endl;
```

Leer hasta fin de línea

```
#include <sstream>
//hacer cin.ignore() antes de getline()
while(getline(cin, line)){
     istringstream is(line);
     while(is >> X)
       cout << X << "␣";
     cout << endl;
}
```

Aleatorios

```
#define RAND(a, b) (rand()%(b-a+1)+a)
srand(time(NULL));
```

Muahaha

```
#include <climits> //INT_MIN, LONG_MAX, ULLONG_MAX, etc.
```

Límites

```
#include <signal.h>
void divzero(int p){
  while(true);}
void segm(int p){
  exit(0);}
//in main
signal(SIGFPE, divzero);
signal(SIGSEGV, segm);
```

Mejorar velocidad

```
ios::sync_with_stdio(false);
```

Leer del teclado

```
freopen("/dev/tty", "a", stdin);
```

File setup

```
//tambien se pueden usar comas: {a, x, m, l}
for i in {a..k}; do cp template.cpp $i.cpp; touch $i.in; done
```